ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

Method for Data Models Integration in the Common Paradigm

Leonid Kalinichenko

Published in collaboration with the
British Computer Society

©Copyright in this paper belongs to the author(s)

ISBN: 3-540-76227-2

Method for Data Models Integration in the Common Paradigm

Leonid Kalinichenko

Institute for Problems of Informatics Russian Academy of Sciences
Moscow, Russia

Abstract

The importance of use of the concept of refinement for heterogeneous multidatabase interoperation research and development is emphasized. Using this concept, the commutative data model mapping method [12] is reconsidered. The method provides for verifiable design of the data model mappings handling the models as formal objects in frame of an abstract metamodel. For such metamodel an abstract machine notation is used combining pure mathematical notation with an ability to automatically prove important properties of the data type definitions as well as the subtype property based on the refinement concept.

The concept of data model refinement is introduced. Data model commutative mapping method is based on this notion. The method is specifically oriented on extensible data model mapping (such as the object data models). To preserve information and operations of types of a specific data model while mapping them into the canonical types the commutativity of two mapping diagrams (data type state and data type behavior diagrams) should be established. The required state-based and behavioral properties of the mappings lead to a proof that a source data model is a refinement of its mapping into the canonical data model.

The applicability of the method is demonstrated on the mapping of the ODMG'93 constructs (the relationship type) into canonical data types. The SYNTHESIS language is used for the canonical data model

¹.

1 Introduction

The notion of shared, interoperable databases is inherently based on integration of multiple data model paradigms, on methodologies of data model translation (mapping) and on an approach for canonical data model (database interlingua) development. The world of DBMSs continues to be extremely diverse. Only for object data models the scale of diversity can be easily perceived from FManola report evaluating OODBMS developments [20] and from X3H7 Object Model Features Matrix [28] intended to be a representative sample of the design space of object models to analyze their interoperability issues.

Industrial projects aiming at the heterogeneous database interoperability continue to be developed. Recently IRO-DB, an ESPRIT III project, focused on integration of heterogeneous object-oriented DBMS and also relational DBMS [9]. The common data model chosen for the project is ODMG'93 [21].

The problem of integration of multiple data models in a single, canonical paradigm remains to be an urgent problem deserving further investigation. In this paper we attempt to emphasize the importance of one of the basic software paradigms, the concept of refinement, for heterogeneous database interoperation research and development.

We assume that completeness of database specifications (up to provision of function specifications in type definitions) is a necessary prerequisite for semantic interoperation [15]. With such level of specification, the subtype definitions should be carefully introduced and checked: to take into account only the relationship of function signatures of type and subtype is not enough. Functions in a subtype should algorithmically refine corresponding functions in a supertype [18].

Some of the related early results published in the area of data model mapping are well known [4, 5, 4, 2, 3, 29, 19, 23, 17].

¹The research described in this publication was made possible in part by the grant INTAS-94-1817 of the INTAS Program

Later a commutative data model mapping approach was developed to implement the heterogeneous multi-database management system [10, 11, 12] in which the strict formal basis for data model transformation and a set of well-defined architectural principles of system design were applied.

Recently a research [26] was undertaken focusing on the need for formal interpretation in which we are able to transform databases between different data models.

Since the development of the mentioned above commutative data model mapping method (oriented on structured data models) object models become dominating [20]. Structured data models motivated the commutative method were not extensible. On the contrary, in the contemporary data models users can easily add their own types with their own behaviors.

Another tightly related development in the same period is the formal software development techniques leading to formation of the mature formal methods possessing the "industrial-strength" (such as B, Raise, VDM, Z) [1, 8, 24]. Importance of these concepts for data model and type specification is well known [27, 18]. Here we exploit these ideas in the interests of data model integration for the database interoperability and reuse.

For this purpose we reconsider a commutative DM mapping method [12] reflecting these major changes. The basic modifications include: 1) adjusting the method itself to the requirements of extensible data models (such as object models); 2) using instead of the denotational semantics of another abstract metamodel combining pure mathematical notation with an ability to automatically prove important properties of the data type definitions (consistency and preservation of invariants by the operations) as well as the subtype property based on the concept of refinement.

The paper is planned as follows. Initially we provide a brief description of the metamodel based on the abstract machines. The metamodel facilities for specification of states and behaviors are defined. Elements of abstract machine notation and their basic modularization features and refinement conditions are introduced. Then we show that the subtyping relation can be interpreted as a refinement of abstract machines. An approach for the heterogeneous data models integration in the common, canonical paradigm is considered in the next section. This approach implies a way of commutative data model mapping construction considered further. Finally an example of constructing of commutative mapping for an ODMG'93 relationship type is presented.

2 Data metamodel

In this research we exploit the abstract machine notation (AMN) [1] for the formal data metamodel elaborating earlier studies [12] where the pure denotational semantics were used for the same purpose. The difficulties with the proofs that we had for the latter [11] are significantly simplified for the AMN due to the existence of the B-Toolkit [1] capable to generate proof obligations and prove them.

AMN as a model-theoretic notation makes possible to integrate the specification of the state space and behavior (specified by operations on the states). Specification of the state of a machine consists in providing of state variables together with invariants - constraints that always should be satisfied. Operations are defined using an extension of Dijkstra's formalism of guarded commands.

State Specification The state specification notation is based on a set theory and a typed first-order language with the built-in types and type (sort) constructors. The collection of complex sort constructors includes: cartesian product ($\times$), powerset ($\wp$), set comprehension ($\{x|x \in s \wedge P\}$), relational sort constructors ($s \leftrightarrow t$), functional sort constructors ($s \longrightarrow t$). Here s, t denote sets, P denotes a predicate. Predicates in the notation are defined using the first-order language. The set of all well-formed formulae (further simply "predicates") is defined using the logical connectors and quantifiers. The interpretation of a state of an abstract machine is given by the machine variable binding assigning to each variable an element in a certain domain.

Specification of Behavior The operations of the abstract machines are based on the generalized substitutions. Every generalized substitution S defines a predicate transformer binding with some postcondition R its *weakest* precondition $[S]R$ that guarantees the invariance of R after an operation execution. If it is so, one says that

S establishes R . "Weakest" precondition means that the "initial state" predicate associated with some given "final state" predicate should allow as many states as possible. The predicate R' is weaker than R iff $R \Rightarrow R'$.

The weakest precondition $[S]R$ is obtained by replacing the variables in R according to the rules provided by different kinds of the generalized substitutions. In complete paper the generalized substitutions of AMN are explained. Elements of the Abstract Machine Notation and basic modularization features of AMN are introduced.

A machine N is said to refine a machine M if a user can use N instead of M without noticing it. Applying algorithmic and data refinement the refinement machine can be constructed [1]. In AMN it is possible to prove formally that a machine is a refinement of another one using specialized tool. Modeling type and subtype by an abstract machine and its refinement we can formally establish the subtyping conditions.

More details on B AMN as the data metamodel will be provided in the complete paper.

3 An approach for heterogeneous data models integration

The basic idea of heterogeneous multidatabase management consists in the introduction of DBMS-independent generalized level of data representation and manipulation — *canonical database level* — and a canonical model of a multidatabase system corresponding to this level. Data models (DM) supported by DBMSs with respect to this canonical DM are *internal* ones.

Formation of a multidatabase system becomes possible on the basis of methods of data model mapping (while speaking about mapping, the DM being mapped will be called a *source* DM and the DM into which the mapping is carried out - a *target* one).

In transition from internal DM to the canonical one it is necessary to preserve the information and operations so that instances of any type t_s defined in a source data model could be used as the instances of a target type t_t mapped from t_s . To reach this we require that the internal DM should *refine* the canonical one. If we reach this, the internal data models will be correctly integrated within the canonical one.

Definition 1. The concept of *data model refinement* is introduced in the following manner.

- Database states in a source and a target DM are *equivalent* iff they are mapped into one and the same state in the content of an *abstract data metamodel*. Such state mapping should be "isomorphic" — i. e. each element of a composite state in one model should correspond to an equivalent element in the other model. It is assumed that equivalent database states represent one and the same collection of facts.
- A type t_s bijectively data refines a type t_t iff the types produce sets of database states of equal power related by bijective dependency in such a way that the states being in one-to-one correspondence are equivalent. For such refinement a data abstraction is a total bijective function relating equivalent states.

In sequel mentioning the type refinement we assume that it possesses bijective data refinement property.

- A schema S_s refines a schema S_t iff for each type t_s of S_s there is a type t_t in S_t (S_t does not contain other types) such that t_s is a refinement of t_t .
- A data model M_s *refines* a data model M_t iff for each admissible schema S_s of M_s there exists an admissible schema S_t of M_t such that S_s is a refinement of S_t .

For the heterogeneous data models integration it is sufficient to require that any source data model should be a refinement of the canonical one: the paradigms of source data models should be integrated within the canonical data model.

3.1 Mapping diagrams

We assume that schemas in any DM consist of definitions of types (in different models we can meet with various notions of types - such as interfaces (ODMG'93), abstract data types (SQL3), classes (ObjectStore), etc.). Taking this into account, we can decompose mapping of schemas into the mapping of type specifications. Even more, in the current analysis we intend to deflect our attention of such peculiarities of object models as inheritance, overriding, metatypes [16]. We focus on a type specification statically released of the relationships in the inheritance, generalization and classification hierarchies of a particular schema. At the same time, we preserve the subtyping relationship between types as well as polymorphism and possibility of late bindings. Thus we can abstract of many peculiarities of specific object models [20] concentrating on clean type specifications and subtyping relationships that we can extract of these models.

For data model M_i the set of all data type schemas expressible in DDL of M_i is denoted by T_i .

Definition 2. The *data type state* corresponding to a type schema $t_i \in T_i$ is a function $s_{t_i} : Id_{t_i} \rightarrow V_i$, defining for each state variable (attribute) of the type schema denoted by identifier $I \in Id_{t_i}$ its value v_i taken from the set of admissible values V_i of the variable type. It is essential that v_i in its turn can also be an analogous function. A set of admissible states corresponding to some type schema $t_i \in T_i$ is a set of functions $S_{t_i} : [Id_{t_i} \rightarrow V_i]$.

A *space of data type states* expressible in M_i is a set of functions $S_i : [Id_i \rightarrow V_i]$, which may be considered as a union of sets $s_{t_i} \in S_{t_i}$ for all $t_i \in T_i$. We consider only admissible states that satisfy the invariants related to types.

Definition 3. The *data type behavior* corresponding to a type schema $t_i \in T_i$ is a function $b_{t_i} : O_{t_i} \rightarrow [S_{t_i} \times S_{t_k} \times \dots \times S_{t_n} \rightarrow S_{t_i}]$ defining for each operation in the type schema the related state transformation for this type. $t_k, \dots, t_n$ are type schemas included into the same database schema that is a type schema t_i . A set of admissible behaviors corresponding to some type schema $t_i \in T_i$ is a set of functions $B_{t_i} : [O_{t_i} \rightarrow [S_{t_i} \times S_{t_k} \times \dots \times S_{t_n} \rightarrow S_{t_i}]]$.

A *space of data type behaviors* expressible in M_i is a set of functions $B_i : [O_i \rightarrow [S_i \times \dots \times S_i \rightarrow S_i]]$, which may be considered as union of sets $b_{t_i} \in B_{t_i}$ for all $t_i \in T_i$.

Definition 4. *Data model* M_i is a triple $\langle T_i, M s_i,$

$M b_i \rangle$ where $M s_i : T_i \rightarrow S_i$ is a semantic state function of M_i , $M b_i : T_i \rightarrow B_i$ is a semantic behavior function of M_i .

Definition 5. The following set of mappings constitutes the mapping f of data model M_j into data model M_i :

- data type schema of M_j into data type schema of M_i mapping:

$$\sigma : T_j \rightarrow T_i,$$

- the data type state space of M_j into the datatype state space of M_i mapping:

$$\theta : S_j \rightarrow S_i,$$

- the data type behavior of M_i into the datatype behavior of M_j mapping:

$$\beta : B_i \rightarrow B_j.$$

3.2 Basic propositions

The following propositions form the basis of a heterogeneous data models integration conception.

Proposition 1. The data model axiomatic extension principle. Canonical data model in the multidatabase management system should be extensible while new source data models are considered. Such extension is implemented axiomatically. Axiomatic in this context means that the extension of a target DM is carried out by addition to its DDL of a system of declarative facilities (axioms) determining (in terms of a target model) logical data dependencies of the source DM. The extension should result in provision of the source DM to be a bijective data refinement of the target data model. Note, that for this step we restrict these extensions only with those DM features that determine the database states. Following axiomatic extension principle, we try to keep the same level of the specification declarativeness in the canonical data model extension as in the source data model.

Construction of a target DM axiomatic extension is considered as a new language design on the basis of the target DM (the canonical DM kernel).

Proposition 2. The data model commutative mapping principle. In the process of mapping the DM of a specific DBMS into a canonical one it is necessary to preserve information and operations. This requirement is satisfied if DM mapping is commutative.

Mapping $f = \langle \sigma, \theta, \beta \rangle$ of data model M_j into extension M_{ij} of data model M_i is commutative iff the following conditions hold:

— data type state diagram is commutative:

— data type behavior diagram is commutative:

— θ mapping is a bijective abstraction function of a data refinement;

— β mapping is an algorithmic refinement.

Here Ω_{ij} denotes a set of axiom schemas expressing the data dependencies of M_j in terms of M_i .

Proposition 3. The unifying canonical data model synthesis principle. Canonical data model synthesis is a process of 1) construction of the canonical data model kernel extensions such that data models of DBMSs embraced by the multidatabase system refine these extensions and 2) merging such extensions in a canonical data model. In such way a unifying canonical data model is formed in which data models of various DBMSs have homogeneous representations (by the subsets of a unifying data model).

3.3 Commutative data model mapping characterization

Axioms $a_{ij} \in \Omega_{ij}$ by which inherent consistency rules fixed in data model M_j and defined in terms of M_i are expressed will be referred to as the data base *invariants*.

For axiomatic extensions of M_i we take the *complete* sets of invariants which define for data types of M_{ij} all consistency rules inherent for M_j .

A data model M_i is *included* into a data model M_j if all axiom schemas Ω_{ri} of an extension M_{ri} of a reference data model M_r such that M_i is its (M_{ri}) refinement are included (possibly not strictly) into the set of axiom schemas Ω_{rj} of an extension M_{rj} of the reference data model M_r such that M_j is its (M_{rj}) refinement.

Proposition 4. Proposition of existence. If the data type state mapping diagram of M_j to M_{ij} commutes, then the behavioral data type mapping diagram can be constructed.

Proposition 5. Proposition of refinement. Data model M_j refines M_i iff there exists commutative mapping of M_j to M_i .

The properties of commutative data model mappings defined above influence greatly the methods of commutative DM mapping design. In particular, from proposition of existence it follows that the process of design of canonical data model kernel axiomatic extensions and the process of design of behavior of types in the extended target data model refined by means of the source one can be separated. Thus, it is allowed to separate and treat independently the process of canonical unifying data model synthesis from the process of definition of the types behavior.

4 Commutative data model mapping construction

The main reason for formal definition of data models is to obtain their compact and precise description, making possible manipulation by different data models as by mathematical objects. We use AMN as the formal data metamodel that is general (i. e. , independent of particular data models concepts), allowing precise expression of semantics properties of different data models, of their similarity or difference on the basis of one and the same language.

Formal definition of data models makes possible the development of a common data model mapping discipline, according to which the construction of data model mapping and of the proof of it's correctness can be done simultaneously.

Constructing the data model mapping, it is natural to use results obtained in the domain of programming languages semantics description. Generally, the definition of programming language L semantics is a mechanism M which relates the syntactical constructions or programs of the language to their content (or denotation — an object corresponding to its own name). In other words $M : P \rightarrow D$, where P is syntactical domain of M (the set of all syntactically correct programs in L), D - semantical domain of M (the set of the program denotations).

Properties of AMN suggest application of compiler-based semantics in which D is the set of target language abstract representation of the syntax of the language (e. g. in the form of a parse tree). It is quite natural to put abstract machines into direct correspondence to data types instead of using another semantics (axiomatic, operational or denotational ones).

Method of commutative data model mapping construction is oriented towards definition of canonical DM kernel extensions that should be refined by the internal data models. Applying compiler-based semantics, we should construct 1) M_j into an extension of M_i mapping; 2) AMN semantics of M_j ; 3) AMN semantics of the extended M_i . After that we can apply AMN technology to prove a) the state-based properties of the mapping (commutativity of the data type state diagrams); b) the behavioral properties of the mapping for all type models defined for a particular internal data model. This leads to a proof that M_j is a refinement of the extension of M_i .

The mapping obtained can be used in process of definition of new, application specific types to get their particular mappings and the respected proofs. Such approach justifies the choice of compiler-based semantics and resembles an idea of computer-assisted application of formal methods for the software design.

Method of unifying canonical data model (UCDM) synthesis. UCDM contains arbitrary data models of different DBMSs reduced to homogeneous refinement-based representations. Process of UCDM synthesis consists of the following stages :

1. source data models set selection:

$$W = \{M_1, M_2, \dots, M_n\};$$

2. setting of partial order on the W set by relation of inclusion of one data model into another ($\leq$);

3. UCDM kernel (M_c) selection. "Minimal" data model M_i , ($i = 1, 2, \dots, n$) is selected so that

$$\forall j_{1 \leq j \leq n} (M_{ri} \leq M_{rj})$$

Here M_{ri} (or M_{rj}) denotes mapping of M_i (or M_j) data models into the fixed reference data model;

4. construction of extensions M_{cj} of the kernel M_c refined by a source model M_j ;

5. UCDM construction as a union of all M_{cj} .

5 Mapping of the ODMG'93 data types into the canonical data types

As an example we consider the mapping of the ODMG'93 relationship type into the SYNTHESIS association metatype.

5.1 ODMG'93 relationship type

ODMG-93 [21] introduces in ODL the concept of binary one-to-one, one-to-many and many-to-many relationships defined between mutable object types. We focus on the one-to-many relationships here. The relationships rediscover CODASYL sets that were well studied in 70ies and 80ies [7]. Semantics of CODASYL sets is dependent on the kind of set membership declared (such as OPTIONAL MANUAL, MANDATORY AUTOMATIC, etc.). Set membership influences the semantics of set member (owner) creation, deletion, modification operations to preserve the referential integrity constraints imposed by the kind of the set membership declared. Partial order can also be imposed on the set type.

The BNF for the ODL relationship specification follows:

```
< relationship spec > ::= [relationship]
    < target of path > < traversal path name >
    < inverse > < inverse traversal path >
    [order_by < attribute list >]
< traversal path name > ::= < string >
< target of path > ::=
    < collection type > < target type >
< target type > ::= < type name >
< inverse traversal path > ::= < target type >
    < traversal path name >
```

Use of the collection type option indicates cardinality greater than one on the target side. If this option is omitted, the cardinality on the target side is one. An ordering criterion is specified with the **order_by** clause. Each attribute used in the ordering criterion must be defined in the property list of the target type definition. Inverse traversal path may be given in both source and target traversal path descriptions.

The operations defined on a one-to-many relationship type are the following:

- create (o1:Denotable_Object,
s:Set<Denotable_Object>)

- delete ()
- add_one_to_one (o1:Denotable_Object,
o2:Denotable_Object)
- remove_one_to_one (o1:Denotable_Object,
o2:Denotable_Object)
- traverse (from:Denotable_Object) ->
s: Set<Denotable_Object>)

We assume that a specific one-to-many relationship R supports rather weak constraint, namely, a partial function $R : T \Rightarrow S$, where T is a set of objects of the target type and S is a set of objects of the source type. The semantics of creation, deletion and modification of objects belonging to T and S should preserve such integrity constraint.

5.2 The related SYNTHESIS language constructs

The SYNTHESIS language [13] provides the *common interoperation platform* for the project ². SYNTHESIS is a multipurpose language. The language should provide for equivalent homogeneous description of various heterogeneous information resources, complete description of the application domains, information system design intended for reuse of the preexisting information resources, application problem definition and solving in HIRE.

The SYNTHESIS object model is defined uniformly with a functional object semantics. The model is functional in the sense that all manipulations of objects are based on the application of functions to objects. Here we focus only on an association metatype of the SYNTHESIS language.

In the language the attribute specifications of objects may be treated in their turn also as types of association objects establishing a correspondence between a set of objects in an association domain and a set of objects in an association range. Thus a specification of an attribute may be considered as a specification of an association type.

Obvious attributes of an association are its domain and range.

Treating an object attribute as an association type motivates an introduction of association metatypes establishing properties of association types. It is said that an object attribute (as an association type) belongs to a particular *attribute category* that is explicitly introduced by an association metatype. Association metatypes have several predefined attributes. A specification of an association metatype is defined as follows:

```
< association metatype >::=
  { < association metatype identifier >;
    in : association, metatype;
    [params : { < formal parameter list >; } ]
    [supertype : < supertype list >; ]
    [inverse : < association metatype identifier >; ]
    [< attribute specification list >; ]
    [instance_section :
      {[association_type : { < bounds >, < bounds >; } ]
      [domain : < domain >; ]
      [range : < range >; ]
      < attribute specification list >; } ]
    }
```

²SYNTHESIS is a research project conducted by the Institute for Problems of Informatics of the Russian Academy of Sciences that is focused on various problems of Heterogeneous Interoperable Information Resource Environment (HIRE) specification, design and management [15]

$\langle \text{bounds} \rangle ::= \{ \langle \text{lower bound} \rangle, \langle \text{upper bound} \rangle \}$
 $\langle \text{lower bound} \rangle ::= \langle \text{arithmetic expression} \rangle \mid \text{inf}$
 $\langle \text{upper bound} \rangle ::= \langle \text{arithmetic expression} \rangle \mid \text{inf}$

Association metatype names (for direct and inverse associations) are used for category names of type attributes. An association type is defined in an instance_section of the association metatype.

An association type is assumed to be a subtype of a set type with elements of a product type defined on an association domain and range types. An association type constraint (defining a kind of binary relation) is set by an attribute association_type. If the association R is defined on a domain C_1 and a range C_2 then the bounds define the following. The first bound gives for any object c_1 of C_1 an admissible range (minimal and maximal value) of a number of different objects of c_2 in C_2 such that $\langle c_1, c_2 \rangle$ belongs to R . The second bound for any c_2 of C_2 gives a minimal and maximal value of a number of objects c_1 of C_1 such that $\langle c_2, c_1 \rangle$ belongs to an association inverse to R . inf is a constant denoting an arbitrary positive integer.

By means of an instance section another attributes of an association type may be defined as usual, thus "attributes of attributes" can be introduced (e.g., for an attribute *price* attributes of this type can be provided, such as *price_status* and *currency*).

In the SYNTHESES language type specifications are syntactically represented by frames, their attributes - by slots of the frames. Additional information related to attributes can be included into metaslots. Syntactically frames are included into figure brackets $\{$ and $\}$, slots are represented as pairs $\langle \text{slot name} \rangle : \langle \text{slot value} \rangle$ (a frame can be used as a slot value), slots in a frame are separated by semi-colons. Metaslots (that are represented by frames) are written immediately after the slots to which they are related. If for a particular object type attribute a metaslot is declared and it is defined that the attribute belongs to a certain attribute category (categories) then a union of the corresponding specifications given in the metaslot and in an instance sections of the corresponding association metatypes is formed.

In the language there exists a possibility of indication that an attribute of a type is an inverse association of another type.

5.3 Mapping of the ODL relationship type into the SYNTHESES data model

We shall map the relationship type into the SYNTHESES association metatype. This case shows how carefully the canonical model kernel should be designed to be extensible. The SYNTHESES association metatype is a loose, unconstrained type that can be easily extended by associating to it new attributes - state, assertional and functional. Thus specific properties of the association types introduced by attribute specifications of ordinary SYNTHESES types can be established.

The ODL relationship type is a concrete, built-in type that might be interpreted as the C++ class for which the specific relationships defined in types may be treated as instances.

Mapping the relationship type into the association metatype, first, we should provide mapping of the type schemas leading to the bijective data refinement of the data type state diagram. To reach that we should provide an appropriate extension of the association metatype introducing adequate axioms.

After that we should provide the appropriate mapping of operations leading to the algorithmic refinement of the data type behavior diagram.

Axiomatic extension of the SYNTHESES association metatype definition providing the ODL relationship to be its data refinement includes two axioms:

1. The axiom of partial functional dependency of the association range on the association domain that we express by fixing the association_type in the metatype definition as follows:
 $\{ \text{association_type} : \{ \{ 0, \text{inf} \}, \{ 0, 1 \} \}$
 and making this metatype attribute obligatory;
2. The axiom of order that is provided by the assertion extending the canonical kernel:
 $\text{ordered_by} : \{ \langle \text{attribute list} \rangle \}$

The latter is optional axiom making association metatype definition to be parameterized with $\langle \text{attribute list} \rangle$. The attribute list is represented as an instance of the sequence of string type. The empty attribute list means that no ordering is implied.

Mapping of operations is provided by specification of the relationship operations create, delete, `add_one_to_one`, `remove_one_to_one` and `traverse` in the attribute specification list of the metatype. The `odl_relationship` type below is an association metatype being the result of the relationship type mapping. Only three operations are shown.

```
{odl_relationship;
in: metatype, association;
params: {attributes: seq_of_string};
inverse: odl_relationship_inv;
instance_section:
  {association_type: {{0, inf},{0, 1}};
  domain: type;
  range: type;
  ord_pred: invariant, {ordered_by: attributes};
  add_one_to_one:{in:function;
    params: {+v1/domain, +v2/range};
    ¬ v2 in this.range & this' =
    union(this, {[v1,v2]})};
  remove_one_to_one:{in:function;
    params: {+v1/domain, +v2/range};
    v1 in this.domain & v2 in this.range &
    this' = differ(this, {[v1,v2]})};
  traverse:{in:function;
    params: {+from/domain, -to/range };
    to' = ordf(attributes, {x/range |
    [from, x] in this } };
```

/ ordf is assumed to be a function that returns a set of objects given by the second parameter ordered by values of ordering attributes given by the first parameter. this means class of associations given by the instance_section.*

Each frame may be declared to belong to some class. Such class membership is given by a slot `in`: $\langle \text{class name list} \rangle$. The meaning of a function parameter sort symbol is: $+$ denotes an input parameter; $-$ denotes an output parameter; $\langle \text{empty} \rangle$ denotes input - output parameter. **/*

```
}}
```

5.4 Using the AMN metamodel to justify the correctness of the mapping

5.4.1 Abstract interpretation of the ODL relationship type in AMN

We introduce two sets - $V1$ and $V2$ interpreting sets of admissible values of two related object types. We consider here only one-to-many relationships and a case when for a collection type in a relationship a set type is used. Abstract machines included into this paper have been developed and proved with the assistance of I.A.Chaban. To interpret relationship we introduce two variables establishing direct and inverse traversal paths in a relationship:

$$\begin{aligned} dpath &\in V1 \mapsto \text{seq}(V2) \\ \wedge ipath &\in V2 \mapsto V1 \end{aligned}$$

Here $\mapsto$ denotes a partial function and $\text{seq}(S)$ is a set of finite sequences of elements from S . Each sequence over a set S is a partial function whose domain is an interval $1..n$ for some natural number n . $dpath$ is considered to be a natural representation of a CODASYL set type that we use here to interpret the relationship. $dpath$ establishes set instances and an order imposed on each of them.

ipath expresses a constraint that we assume to be imposed by an ODMG relationship corresponding to CODASYL sets having OPTIONAL MANUAL membership. The following invariants interrelates *dpath* and *ipath*:

$$\begin{aligned} \text{dom}(dpath) &= \text{ran}(ipath) \\ \wedge \forall xx. (xx \in \text{dom}(dpath) \Rightarrow \text{ran}(dpath(xx)) \\ &= ipath^{-1}[\{xx\}]) \end{aligned}$$

Here *dom* and *ran* are a domain and a range of a binary relation, r^{-1} is an inverse of a relation formed by interchanging the elements of each ordered pair constituting the relationship. To interpret ordered relationships we introduce the type

$$\begin{aligned} fsort &\in \wp(ATTRLIST) \times \wp_1(VV) \longrightarrow \\ &\text{seq}(VV) \\ \wedge \forall (xx, yy). (xx \subseteq ATTRLIST \\ \wedge yy \in \wp_1(VV) \\ \Rightarrow fsort(xx \mapsto yy) \in \text{perm}(yy)) \end{aligned}$$

where $\wp_1(S)$ denotes a set of all non-empty subsets of S , $\text{perm}(S)$ is a set of bijective sequences of elements of a finite set S : $\text{perm}(S) = 1..card(S) \longleftrightarrow S$, where $\longleftrightarrow$ denotes a set of bijections. $(E \mapsto F)$ denotes in AMN an ordered pair. Ordering is simulated by the operation *order* of the abstract machine *ord*:

MACHINE

ord (*VV* , *ATTRLIST*)

VARIABLES *fsort*

INVARIANT

$$\begin{aligned} fsort &\in \wp(ATTRLIST) \times \wp_1(VV) \longrightarrow \\ &\text{seq}(VV) \\ \wedge \forall (xx, yy). (xx \in \wp(ATTRLIST) \wedge yy \in \\ \wp_1(VV) \Rightarrow fsort(xx \mapsto yy) \in \text{perm}(yy)) \end{aligned}$$

ASSERTIONS

$$\begin{aligned} \forall (xx, yy). (xx \in \wp(ATTRLIST) \wedge yy \in \\ \wp_1(VV) \wedge fsort(xx \mapsto yy) \in \text{perm}(yy) \Rightarrow \\ \text{ran}(fsort(xx \mapsto yy)) = yy) \end{aligned}$$

/* **ASSERTIONS** is a list of predicates separated by $\wedge$ which gives properties that can be asserted from the machine invariant and other contextual information. */

OPERATIONS

$$oo \leftarrow \text{order}(par1, par2) =$$

PRE

$$\begin{aligned} par1 &\subseteq VV \wedge \\ par2 &\subseteq ATTRLIST \end{aligned}$$

THEN

$$oo := fsort(par2 \mapsto par1)$$

END

/* **PRE P THEN S END** is the AMN equivalent for the pre-conditioned substitution $P \mid S$. */

END

Complete definition of an abstract machine interpreting the ODMG relationship type as a refinement of the corresponding canonical type is given in a separate subsection below.

5.4.2 Abstract interpretation of the extended SYNTHESIS association metatype in AMN

As a loose, unconstrained type the SYNTHESIS association metatype is naturally interpreted in AMN by a very general form of binary relation of AMN:

$$rr \in V1 \leftrightarrow V2$$

equivalent to powerset $\wp(V1 \times V2)$. Now we shall impose additional constraints corresponding to two axioms extending an association metatype to make the ODMG relationship type its bijective data refinement. To the first axiom

$$\{\text{association_type} : \{\{0, \text{inf}\}, \{0, 1\}\}$$

the following invariant of AMN corresponds:

$$\forall xx. (xx \in \text{dom}(rr) \Rightarrow \text{card}(rr[\{xx\}]) \geq 1)$$

Here $rr[s]$ is an image of set s under rr . To interpret an axiom of order we impose an ordering constraint similarly to the ODMG type:

$$\text{ordf} \in \wp_1(ATTR_V2) \times \wp_1(V2) \longrightarrow \text{seq}(V2)$$

$$\wedge \forall (xx, yy). (xx \in \wp_1(ATTR_V2) \wedge yy \in$$

$$\wp_1(V2) \Rightarrow \text{ordf}(xx \mapsto yy) \in \text{perm}(yy))$$

The complete AM corresponding to the association metatype looks as follows. For brevity we include only three operations of the machine: *add_one_to_one*, *remove_one_to_one* and *traverse*.

MACHINE

canonical_rel

SETS

$V1 ; V2 ; ATTR_V2$

VARIABLES

$rr, \text{ordf}, \text{attr_v2}$

INVARIANT

$$rr \in V1 \leftrightarrow V2$$

$$\wedge \forall xx. (xx \in \text{ran}(rr) \Rightarrow$$

$$\text{card}(rr^{-1}[\{xx\}]) = 1)$$

$$\wedge \text{ordf} \in \wp(ATTR_V2) \times \wp_1(V2) \longrightarrow$$

$$\text{seq}(V2)$$

$$\wedge \forall (xx, yy). (xx \in \wp(ATTR_V2) \wedge yy \in$$

$$\wp_1(V2) \Rightarrow \text{ordf}(xx \mapsto yy) \in \text{perm}(yy))$$

$$\wedge \text{attr_v2} \in \wp(ATTR_V2)$$

ASSERTIONS

$$\forall (xx, yy). (xx \in \wp(ATTR_V2)$$

$$\wedge \text{ordf}(xx \mapsto yy) \in \text{perm}(yy) \Rightarrow$$

$$\text{ran}(\text{ordf}(xx \mapsto yy)) = yy)$$

OPERATIONS

$$\text{add_one_to_one}(par1, par2) =$$

$$\text{PRE } par1 \in V1 \wedge par2 \in V2$$

$$\wedge \neg (par2 \in \text{ran}(rr))$$

THEN

$$rr := rr \cup \{par1 \mapsto par2\}$$

END ;

$$\text{remove_one_to_one}(par1, par2) =$$

$$\text{PRE } par1 \in \text{dom}(rr)$$

```

 $\wedge par2 \in rr [ \{ par1 \} ]$ 
THEN
 $rr := rr - \{ par1 \mapsto par2 \}$ 
END;

 $to \leftarrow traverse( from ) =$ 
PRE  $from \in \text{dom}( rr )$ 
THEN  $to := \text{ordf} ( attr\_v2 \mapsto rr [ \{ from \} ] )$ 
END
END

```

5.4.3 The refinement machine

To define the AM corresponding to the ODMG relationship type as a refinement of the association metatype machine we establish the following abstraction bijective relation:

```

 $\text{dom}( dpath ) = \text{dom}( rr )$ 
 $\wedge \forall xx . ( xx \in \text{dom}( dpath ) \Rightarrow \text{ran}( dpath ( xx ) )$ 
 $= rr [ \{ xx \} ] )$ 
 $\wedge fsort = \text{ordf}$ 
 $\wedge ipath = rr^{-1}$ 

```

Now we can introduce the definition of the AM being a refinement of the extended association metatype in SYNTHESIS by the ODMG relationship type:

```

REFINEMENT
 $odmg\_rel$ 
REFINES
 $canonical\_rel$ 
INCLUDES
 $ord ( V2 , ATTR\_V2 )$ 
VARIABLES
 $dpath , ipath , attr\_pp2$ 
INVARIANT
 $dpath \in V1 \mapsto \text{seq}( V2 )$ 
 $\wedge ipath \in V2 \mapsto V1$ 
 $\wedge \text{dom}( dpath )$ 
 $= \text{ran}( ipath )$ 
 $\wedge \forall xx . ( xx \in \text{dom}( dpath ) \Rightarrow \text{ran}( dpath ( xx ) )$ 
 $= ipath^{-1} [ \{ xx \} ] )$ 
 $\wedge \forall xx . ( xx \in \text{dom}( dpath ) \Rightarrow$ 
 $dpath ( xx ) = fsort ( attr\_pp2 \mapsto$ 
 $\text{ran}( dpath ( xx ) ) ) )$ 
 $\wedge attr\_pp2 \in \wp ( ATTR\_V2 )$ 
 $\wedge attr\_pp2 = attr\_v2$ 
 $\wedge \text{dom}( dpath ) = \text{dom}( rr )$ 
 $\wedge \forall xx . ( xx \in \text{dom}( dpath ) \Rightarrow \text{ran}( dpath ( xx ) )$ 
 $= rr [ \{ xx \} ] )$ 
 $\wedge fsort = \text{ordf}$ 
 $\wedge ipath = rr^{-1}$ 

```

OPERATIONS

add_one_to_one(*par1* , *par2*) =

PRE *par1* $\in V1 \wedge par2 \in V2$

$\wedge \neg (par2 \in \text{dom}(ipath))$

THEN

IF *par1* $\in \text{dom}(dpath)$

THEN

ANY *pp* **WHERE** *pp* $\in \text{seq}(V2)$ **THEN**

pp $\leftarrow \text{order}(\text{ran}(dpath(par1)))$

$\cup \{par2\}, attr_pp2$;

dpath(*par1*) := *pp*

END

ELSE

ANY *pp* **WHERE** *pp* $\in \text{seq}(V2)$ **THEN**

pp $\leftarrow \text{order}(\{par2\}, attr_pp2)$;

dpath(*par1*) := *pp*

END

END ;

ipath(*par2*) := *par1*

END ;

/* **ANY** *xx* **WHERE** *P* **THEN** *S* **END** is the AMN equivalent for generalized substitution @*xx*.(*P* $\implies$ *S*). */

remove_one_to_one(*par1* , *par2*) =

PRE *par1* $\in \text{dom}(dpath) \wedge par2 \in$

$\text{ran}(dpath(par1))$

THEN

IF $\text{ran}(dpath(par1)) - \{par2\} = \emptyset$

THEN *dpath* := $\{par1\} \trianglelefteq dpath$

/* *s* $\trianglelefteq$ *r* is antirestriction of *r* by *s*. Here *r* $\in S \leftrightarrow T$ and *s* $\in S$. The antirestriction gives the set: $\{x, y \mid x, y \in r \wedge x \in S - s\}$. */

ELSE

LET *qq* **BE** *qq* = $\text{ran}(dpath(par1)) - \{par2\}$

IN

ANY *pp* **WHERE** *pp* $\in \text{seq}(V2)$ **THEN**

pp $\leftarrow \text{order}(qq, attr_pp2)$;

dpath(*par1*) := *pp*

END

END

END ;

ipath := *ipath* - $\{par2 \mapsto par1\}$

END;

/* **LET** *xx* **BE** *xx* = *E* **IN** *S* **END** is the AMN equivalent for the generalized substitution @*xx*.(*xx* = *E* $\implies$ *S*). */

to $\leftarrow \text{traverse}(\text{from}) =$

PRE *from* $\in \text{dom}(dpath)$

THEN *to* := *dpath*(*from*)

END

END

AMN specifications above were used for generation of the refinement proof obligations and formal proofs justifying correctness of the mapping constructed. During this process several errors in specifications were corrected. The errors usually were related to some border conditions in the abstract machine and the refinement that were not defined consistently.

5.5 An example of the relationship type mapping

We illustrate here mapping of two interfaces defined in ODL into the SYNTHESIS types emphasizing the relationship type mapping. The definitions of the Professor and Section interfaces in ODL looks as follows (an abridged example is taken from [21]):

```
interface Professor: Employee {
    extent professors;
    relationship Set<Section> teaches
    inverse Section::is_taught_by
}

interface Section {
    extent sections;
    relationship Professor is_taught_by
    inverse Professor::teaches
}
```

The mapping of these definitions into SYNTHESIS gives two type declarations. The attribute *teaches* of the type *Professor* is declared to be an instance of the *odl_relationship* metatype together with the conjugate inverse attribute *taught_by* of the type *Section*.

```
{Section;
in: type;
instance_section:
{taught_by: Professor;
metaslot
in: odl_relationship_inv;
inverse: Professor.teaches;
end}
}

{Professor;
in: type;
instance_section:
{teaches: set_of Section;
metaslot
in: odl_relationship;
inverse: Section.taught_by;
end}
}
```

6 Conclusion

The paper contributes to the creation of a methodological basis for solving the problem of data model heterogeneity in multidatabase interoperable environment. Using the concept of data model refinement, the commutative data model mapping method [11, 12] is reconsidered. The method provides for verifiable design of the data model mappings handling the models as formal objects in frame of an abstract metamodel. For such

metamodel an abstract machine notation [1] is used combining pure mathematical notation with an ability to automatically prove important properties of the data type definitions as well as the subtype property based on the refinement concept.

The data model axiomatic extension principle, the data model commutative mapping principle, the unifying canonical data model synthesis principle based on the notion of *data model refinement* are exploited to create the necessary ground for the design of semantically interoperable heterogeneous multidatabase environment. Canonical data model integrating various paradigms of heterogeneous source data models is the basic constituent of such environment.

According to the approach for commutative data model mapping construction the basic steps of the mapping design should:

- construct the mapping of a source data model type specifications into type specifications of an extension of the canonical data model (including state and behavior mapping);
- provide an interpretation of source data model types in abstract machine notation;
- provide an interpretation in abstract machine notation of the types resulted in mapping of the source data model types into an extension of the canonical data model;
- justify the state-based and behavioral properties of the type mappings proving that a source data type is a refinement of its mapping to the canonical data model.

It is important that the process of design of canonical data model kernel axiomatic extensions (state mapping diagram construction) and the process of design of behavior of types in the extended target data model refined by means of the source one (behavior mapping diagram construction) can be separated. Thus, it is allowed to separate and treat independently the process of the canonical unifying data model synthesis from the process of the definition of the types behavior.

The spectrum of data models embraced by the heterogeneous multidatabase environment may include object-oriented as well as structured data models. A unifying canonical data model may be synthesized integrating data models of various DBMSs on the basis of the data model refinement conception. The canonical data model with an object-oriented kernel can be used.

References

- [1] Abrial J.-R. B-Technology. Technical overview. BP International Ltd., 1992, 73 p.
- [2] Biller H., Glatthaar W. On the semantics of data bases: the semantics of data definition language, Institut f. Informatik, Univesity of Stuttgart, Nov. 1975, p. 1-19.
- [3] Biller H., Glatthaar W., Neuhold E.J. On the semantics of data bases : the semantics of data manipulation languages. - In Proc. IFIP TC-2 Working Conference on Modelling in DBMS. Amsterdam: North-Holland, 1976, p. 239-267.
- [4] Biller H., Neuhold E.J. Semantics of data bases: the semantics of data models. Information systems, vol. 3, 1978, p. 11 - 30.
- [5] Borkin S.A. Data model equivalence. Proc. of the 4-th International Conference on VLDB, West Berlin, N-Y.:IEEE, 1978 , p. 526 - 534.
- [6] Enrich H.-D., Engels G., Paredaens J., and Wegner P. Fundamentals of Object-Oriented Languages, Systems and Methods. Dagstuhl Seminar Report, 95, August 1994.
- [7] CODASYL Data Description Language Committee Journal of Development. January 1978

- [8] FME'93: Industrial-Strength Formal Methods. Proceedings of the First International Symposium of Formal Method Europe, Odense, Denmark, April 1993, LNCS 670, Springer- Verlag, 1993
- [9] Huck G., Fankhauser P., Busse R., Klas W. IRO- DB: an object-oriented approach towards federated and interoperable DBMS. Proceedings of the International Workshop "Advances in Databases and Information Systems", Moscow, 1994
- [10] Kalinichenko L.A. Data model transformation method based on axiomatic data model extension.-In Proc. of the 4-th International Conference on VLDB. West Berlin, N-Y.:IEEE, 1978.
- [11] Kalinichenko L.A. Methods and Tools for Integration of Heterogeneous Databases.-Moscow.:Science Publ., 1983, 423p. (in Russian).
- [12] Kalinichenko L.A. Methods and tools for equivalent data model mapping construction. Proc. of the EDBT'90 Conference,1990,Springer Verlag,p.92-119.
- [13] Kalinichenko L.A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment.- Institute for Problems of Informatics, Russian Academy of Sciences, Moscow, 1995, 103 p.
- [14] Kalinichenko L.A. Properties of commutative data model mappings.- Cybernetics, N5, 1985, p. 37-40 (in Russian).
- [15] Kalinichenko L.A. Emerging semantic-based interoperable information system technology. Computers as our better partners. Proceedings of the International IISF/ACM Symposium, Tokyo, World Scientific, 1994
- [16] Klas W. A metaclass systems for open object-oriented data models. Doctoral thesis, Vienna TU, January 1990.
- [17] Klug A.C. Theory of database mappings. Technical report CSRG-98, Dec. 1978, University of Toronto.
- [18] Liskov B., Wing J.M. Specifications and their use in defining subtypes. OOPSLA'93
- [19] Maibaum T.S. Mathematical semantics and a model for data bases. Information Processing 77, North Holland, 1977, p. 133 - 138.
- [20] Manola F. An evaluation of Object-Oriented DBMS Developments GTE Laboratories, Waltham, MA, TR-0263-08-94- 165, August 1994
- [21] The Object Database Standard: ODMG-93. Ed. by R.G.G. Cattell, Morgan Kaufmann Publ., 1994, p. 169.
- [22] Object Management Group, "The Common Object Request Broker: Architecture and Specification", OMG Document Number 91.12.1, December 1991.
- [23] Paolini P., Pelagatti G. Formal definition of mappings in a data base.- ACM Sigmod Proc., August 1977, p. 40-46.
- [24] Spivey J.M. The Z Notation. A reference manual. Prentice-Hall, 1989
- [25] Fraser M.D., Kumar K., Vaishnavi V.K. Strategies for incorporating formal specifications in software development. CACM, v. 37, N 10, October 1994
- [26] De Troyer O.M.F. On data schema transformation. Ph. D. Thesis, Brabant Univ., 1993
- [27] Wetzel I., Schmidt J.W., Mertikas M., Borgida A. Apecification and refinement of databases and transactions. DAIDA Deliverable, MAP2.1, April 1990
- [28] X3H7 Object Model Features Matrix, X3H7-93-007v10, February, 1995
- [29] Zaniolo C. Design of Relational Views over Network Schemas. SIGMOD 1979 Conference Proceedings, May 1979, p. 179- 189.